Dynamic Programming algorithm for Computing Temporal Logic Robustness

by

Hengyi Yang

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2013 by the
Graduate Supervisory Committee:

Georgios Fainekos, Chair
Hessam Sarjoughian
Aviral Shrivastava

ARIZONA STATE UNIVERSITY

May 2013

ABSTRACT

In this thesis we deal with the problem of temporal logic robustness estimation. We present a dynamic programming algorithm for the robust estimation problem of Metric Temporal Logic (MTL) formulas regarding a finite trace of time stated sequence. This algorithm not only tests if the MTL specification is satisfied by the given input which is a finite system trajectory, but also quantifies to what extend does the sequence satisfies or violates the MTL specification. The implementation of the algorithm is the DP-TALIRO toolbox for MATLAB. Currently it is used as the temporal logic robust computing engine of S-TALIRO which is a tool for MATLAB searching for trajectories of minimal robustness in Simulink/ Stateflow. DP-TALIRO is expected to have near linear running time and constant memory requirement depending on the structure of the MTL formula. DP-TALIRO toolbox also integrates new features not supported in its ancestor FW-TALIRO such as parameter replacement, most related iteration and most related predicate. A derivative of DP-TALIRO which is DP-T-TALIRO is also addressed in this thesis which applies dynamic programming algorithm for time robustness computation. We test the running time of DP-TALIRO and compare it with FW-TALIRO. Finally, we present an application where DP-TALIRO is used as the robustness computation core of S-TALIRO for a parameter estimation problem.

ACKNOWLEDGEMENTS

I want to thank Dr. Georgios Fainekos for this wonderful opportunity to work on this interesting research topic and more importantly work along with him. I benefit greatly from his guidance, rigorous attitude and forward-looking spirit. I appreciate the financial support from Dr. Georgios Fainekos and Arizona State University in the past two years. I would also like to thank Dr. Hessam Sarjoughian and Dr. Aviral.Shrivastava for the support and feedback they gave me as part of my thesis committee.

I want to thank my friends and colleagues at Arizona State University. I would particularly like to thank colleagues of CPS lab including Parth Pandya, Ramtin Kermani, Shihkai Su, Kangjin Kim, Shashank Srinivas, Bardh Hoxha, and Adel Dokhanchi who helped and inspired me during past few years.

Also, I want to thank all the professors and staffs of Arizona State University who altogether create a great learning and working environment and atmosphere for me.

This work has been partially supported by NSF award CNS-1017074. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation of the Thesis

Nowadays the use of cyber-physical system (CPS) can be found in a wide range of applications including automotive, aerospace, healthcare, transportation, infrastructure, military and so on. The so called CPS represents a combinatorial system of computation, networking and physical elements. While traditional embedded systems are designed to achieve specific goals independently, most CPSs are designed with feedback loops of which the inputs and outputs from physical elements would affect the final computation result of the whole system and vice versa. The potential of such system has been recognized gradually, and investments are made worldwide in developing the technology. With the trend of CPSs being more diverse and universal in everyday life and especially because of the critical areas where CPSs are used, it is essential to ensure correctness, security and reliability of such systems and software deployed. We have already paid extremely high price for software failures in the past. For example, the unmanned rocket Ariane 5 Disaster [8] in 1996 which lead to a loss of more than 370 million dollars. Ariane 5 explored 40 seconds after its launching due to software error. However the same program functioned perfectly on Ariane 4, the only change had been made is the physical part of the rocket. Thus, the need of system verification and validation is crucial for CPSs.

Model Checking [25] is a tool that is very useful for verification of both software and hardware systems and it has got increased attention from academia as well as industries

of automobiles and avionics. Model checking works as follows: Engineers establish a model of a system by abstracting the dynamical characteristics of a physical object or a set of physical parts, such as internal combustion engine and transmission gearbox, with mathematical and logical models. And the model needs to be checked with some specifications automatically. However, model checking is only suitable for finite-state systems. It does not apply to systems with infinite state space including continuous systems and hybrid systems. In some cases, model checking problem is undecidable for systems whose state space is infinite space [26]. Recently, progress has been made to use temporal logic to capture more information and better express the characteristics of continuous and discrete-time signals. In this thesis, we mainly focus on Metric Temporal Logic (MTL) [3] which provides the ability to express the time-varying behaviors of continuous and hybrid systems.

One of the main motivations of the work in this thesis is the great efforts done by Fainekos and Pappas [2] to apply robustness interpretation of MTL for continuous-time signals in metric spaces. One can obtain not only traditional Boolean value of satisfiability, but also the degree of how far away the specification is satisfied of falsified. It is very useful application-wise especially in the optimization setting of a control function to manage the behavior of the model of a physical system in [2]. The computation of temporal logic robustness was implemented in a MATLAB toolbox called FW-TALIRO which is one of the building blocks of the overall framework called S-TALIRO [21]. Both toolboxes are available at [27].

However, industrial-scale systems can be quite complex. A system model can contain as much as thousands of blocks or complicated hierarchical structure with lookup tables and shared variables. The MTL formula may have dozens of predicates and temporal logical operators with all kinds of time constraints and the real-time trace can be multi-dimension with even millions of timed states. We found that FW-TALIRO does not scale well under such circumstances. In fact, it may take several minutes for FW-TALIRO to compute one robustness metric of a reasonable-size MTL formula over a real-value trace with ten-thousand timed states. It is almost impossible to use FW-TALIRO for optimization or falsification problem of these kinds of system since they usually require hundreds of robustness metric computations. In this thesis, we propose an improved algorithm to address the excessive time of computing robustness metric.

Example 1.1.1: As an motivating example, we present a parameter estimation problem of a Simulink model for a four-speed automatic transmission [12, 13, 15] of a vehicle as shown in Fig 1.1. This example is presented in [11]. The model has two inputs: the percentage of throttle schedule and brake schedule. And the output is the RPM of the engine and the speed of the vehicle. In this example, we set the brake schedule to 0 for 30 seconds. The throttle schedule at each point in time can be any value from 0 (fully closed) to 100 (fully open). At time 0 the vehicle is still so the speed and RPM is 0 initially. We are interested in solving problems such as "What is the maximum time that the RPM ω cannot exceed 4500 whatsoever".

As demonstrated in [11], this time estimation problem can be posed as a parameter estimation problem in an MTL formula. Moreover, the parameter estimation problem is further reduced to an optimization problem where the cost function is the MTL robustness. This optimization problem is solved using stochastic search techniques and, thus, the robustness computation must be performed quickly.

FW-TALIRO is not suitable to solve such problems due to the large number of robustness values needed to be computed and high running time of FW-TALIRO. Thus we need to develop a replacement that has much improved performance over FW-TALIRO. We came up with the solution to apply dynamic programming algorithm for temporal logic robustness and implemented in DP-TALIRO toolbox.

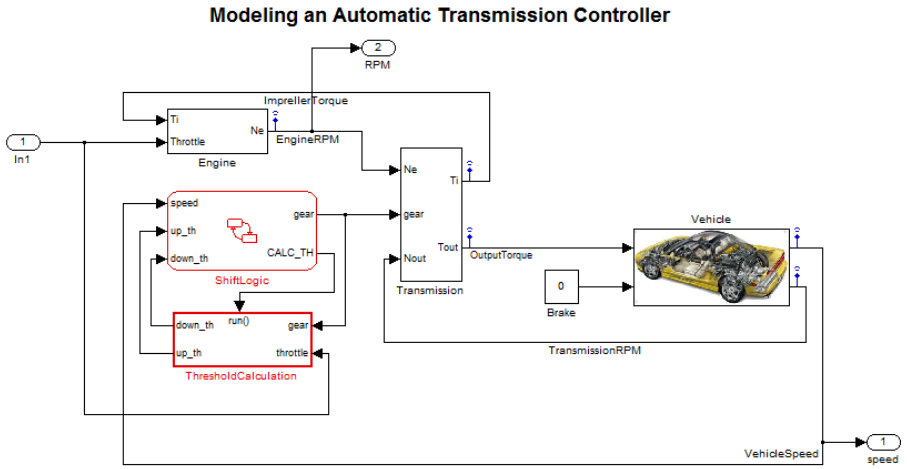

Fig 1.1: The Simulink model of an automatic transmission controller

1.2 Contribution of the Thesis

The main contribution of this thesis is that we have refined the dynamic programming algorithm for temporal logic robustness problem and we implement the algorithm into DP-TALIRO toolbox. As opposed to FW-TALIRO which uses formula rewriting techniques, DP-TALIRO shows tremendous improvement on the running time and memory used which as a result allows DP-TALIRO to handle larger size of input sequences and more complex specifications.

DP-TALIRO also integrates features such as dynamic programming algorithm for polarity and parameter estimation [10]. The implementation details are provided in this thesis as well. We also present the algorithm for DP-T-TALIRO which is the toolbox for dynamic computing time robustness [9] as a derivative of DP-TALIRO. It is also integrated in S-TALIRO and it can run as a stand along toolbox as well.

1.3 Thesis Structure

This thesis is structured according to the following outline:

- Chapter 1: The first chapter introduces the motivation of the thesis.
- Chapter 2: In this chapter, we present the background and fundamentals of the work including Metric Temporal Logic, definition of robustness and time robustness as well as related researches.
- Chapter 3: In this chapter, we present the dynamic programming algorithm along with implementation details and details of other features incorporated in the toolbox.

- Chapter 4: In this chapter, we analyze the running time or DP-TALIRO and compare it with FW-TALIRO.
- Chapter 5: We present an application where DP-TALIRO is used as the robustness computation core of S-TALIRO for a parameter estimation problem.
- Chapter 6: In the final chapter we make a conclusion and discuss some possible future work.
- Appendices: The Appendices include the user manual of DP-TALIRO and DP-T-TALIRO.

Chapter 2

BACKGROUND AND FUNDAMENTALS

2.1 Metric Temporal Logic

First, we recap the syntax and semantics of Metric Temporal Logic (MTL) here. MTL is originally defined in [3]. Given a finite set AP of atomic propositions, the MTL formula is defined recursively by time-constrained temporal operators as follows:

$$\varphi \vDash \text{true} \mid p \mid \varphi_1 \wedge \varphi_2 \mid \neg\, \varphi_1 \mid\ \varphi_1 \mathrm{U}_I \varphi_2\ \mid \mathrm{X}\varphi_1$$

Where p∈AP and ***I*** could be an open, closed or half-open half-closed interval whose left and right end-points are rational numbers or ∞.If *I* equals to [0,+ ∞) then *I* is omitted in the notation.

The MTL formula supports standard propositional constants and operators: true, false, and (∧), or (∨), indicate (→), equivalent (↔) as well as temporal logic operators such as always (G), eventually (F), until (U) and release (R). 'Eventually', 'always' and 'release' can be derived from 'until'.

$$\mathrm{F}_I \varphi = \text{true}\ \mathrm{U}_I\ \varphi$$

$$\mathrm{G}_I \varphi = \neg\ \mathrm{F}_I \neg\ \varphi.$$

$$\varphi_1 \mathrm{R}_I \varphi_2 = \neg\ (\neg\ \varphi_1 \mathrm{U}_I \neg\ \varphi_2)$$

Then, we define satisfaction problem. Given a trace which is a timed state sequence $s = (\sigma, \tau)$ where σ denotes a finite sequence of states and τ is the timestamps which is a finite sequence of real numbers and $|\sigma| = |\tau|$ meaning the length of σ and τ are the

same. Intuitively, a sequence s represents an execution of a system model. We can interpret a sequence s as at time τ_i the system was in state σ_i.

We define the notation $\mathcal{O}$ as an observation map so that $\mathcal{O}(p)$ represents the set of p. The formal definition is $\mathcal{O}$: AP $\rightarrow$ P(X) such that for every $p \in AP$ we have the corresponding set $\mathcal{O}(p) \subseteq X$.

We define that a trace s at time τ_i satisfies a formula φ, written as $(s, i) \vDash \varphi$ inductively over the structure of the MTL formula as follows:

$(s, i) \vDash$ true	is always true;
$(s, i) \vDash$ false	is always false;
$(s, i) \vDash p$	iff $\sigma_i \in \mathcal{O}(p)$;
$(s, i) \vDash \varphi_1 \wedge \varphi_2$	iff $(s, i) \vDash \varphi_1$ and $(s, i) \vDash \varphi_2$;
$(s, i) \vDash \varphi_1 \vee \varphi_2$	iff $(s, i) \vDash \varphi_1$ or $(s, i) \vDash \varphi_2$;
$(s, i) \vDash \neg\varphi_1$	iff $(s, i) \nvDash \varphi_1$;
$(s, i) \vDash G_I\varphi_1$	iff for all $i \in I$ that $(s, i) \vDash \varphi_1$;
$(s, i) \vDash F_I\varphi_1$	iff exist $i \in I$ that $(s, i) \vDash \varphi_1$;
$(s, i) \vDash \varphi_1 U_I \varphi_2$	iff exist $j \geq i$ while $(s, j) \vDash \varphi_2$ with $j \in i + I$ and for all $j > k \geq i$ $(s, k) \vDash \varphi_1$;

2.2 Robustness

2.2.1 Space robustness

By contrast with traditional use of temporal logic which a verdict evaluates to whether a certain trace meets or violates an MTL formula. In this thesis we use the concept of robustness degree for finite timed state sequences as introduced in [2]. The robustness degree expresses how much error the signal could tolerate or how far away is the signal to meet certain specification. Note that since the robustness here is essentially the distance between the given signal and the boundary of the set of signals that satisfy the requirement, we denote the robustness mentioned here as space robustness, as opposed to time robustness that we will introduce later. In the rest of the paper, when we say 'robustness' we mean space robustness and we will specifically say 'time robustness' if we need to use time robustness.

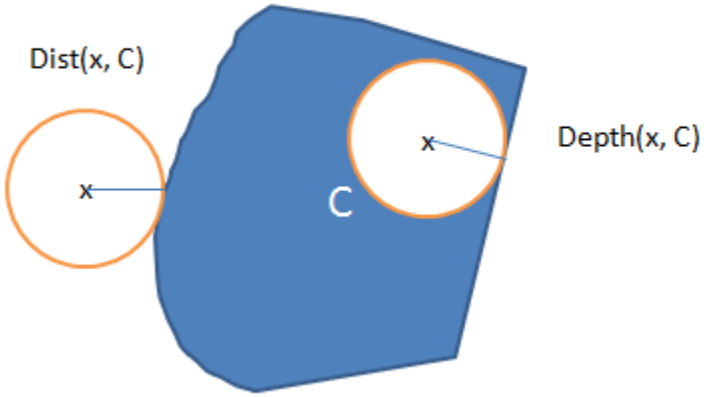

Fig 2.1: The definition of distance and depth

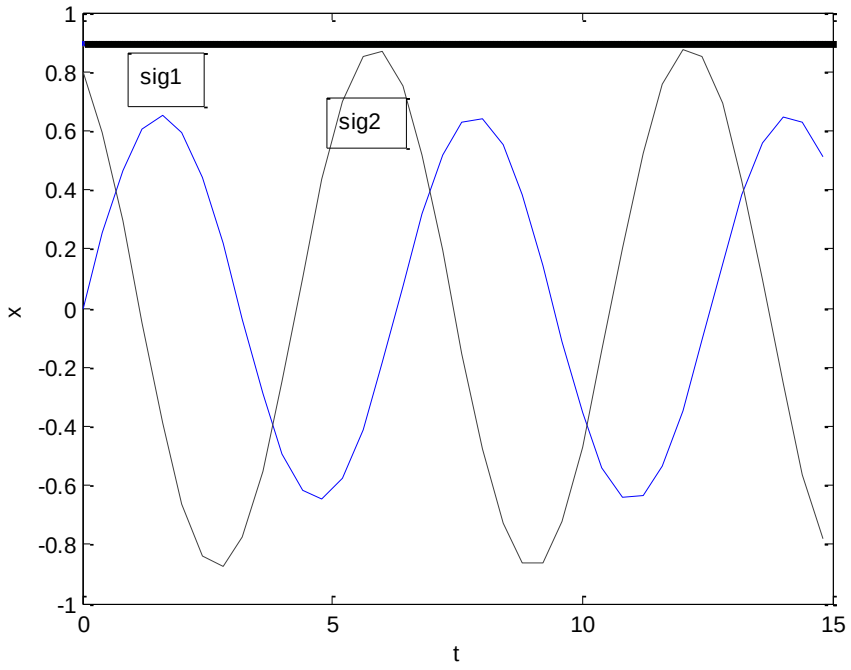

Fig 2.2: Two signals sig1 and sig2 satisfy the specification G(x<0.9)

Why we need space robustness? Here we present an example illustrating the importance of space robustness. As shown in Fig 2.1, the distance defined is the shortest distance from the point to any points inside set C. Similarly the depth defined is the shortest distance from the point to any points outside set C. Thus this value defines how robust certain signal is according to certain specification expressed by MTL. For instance, consider sig1 and sig2 in Fig 2.2. The specification here requires that the input signal should always be less than 0.9. Even though sig1 and sig2 both meet this requirement, obviously sig1 meets the specification by a good margin while sig2 barely meets the specification and sig1 would have the better ability to resist noise interferences than sig2. Thus with the notion of robustness degree we would be able to capture this kind of characteristic of signals.

Furthermore, the solution given in this thesis would not merely consider space robustness, but also time robustness as defined in [9]. Time robustness defines how robust an MTL formula is regarding a sequence at a certain point in time. In a nutshell time robustness indicates how faraway the trace could shift in time to the future or to the past without changing the satisfaction or violation status of an MTL formula.

The formal definition of space robustness degree is given as follows. Here, the distance and depth notion is based on the generalized metric d. We refer reader to [4] regarding the details of generalized metric.

First we define the distance from a point to a set. Let X and C be two sets, $C \subseteq X$, x be a point and $x \in X$, and d be a metric that induce the topology on set X. Then we could define the Signed Distance from point x to set C to be:

$$\mathrm{Dist}_d(x, S) := \begin{cases} -\mathrm{dist}_d(x, C), & \text{if } x \notin C \\ \mathrm{depth}_d(x, C), & \text{if } x \in C \end{cases}$$

While $\mathrm{dist}_d(x, C) := \inf\{d(x, y) \mid y \in C\}$;

$$\mathrm{depth}_d(x, C) := \mathrm{dist}_d(x, X \backslash C) \ ;$$

That is to say this distance defined is the shortest distance from the point to any points inside set C. Similarly the depth defined is the shortest distance from the point to any points outside set C. Thus this value defines how robust a certain signal is according to certain specification expressed by MTL. Noted that here we use the extended definition regarding supremum (⊔) and infimum (⊓). To be specific, we define the supremum of empty set to be the smallest element of the domain and the infimum of empty set to be the largest element of the domain.

2.2.2 Discrete-Time Robust Semantics

In this section we combine MTL formula and robustness notion [15]. Here we introduce the semantics which maps a discrete-time trace s regarding an MTL formula φ to a value from a partially ordered set V. We denote the robustness value of formula φ regarding trace s at sampling point *i* by $[\varphi]_d(s,i)$. We define $\tau^{-1}(R) = \{i \in \mathrm{N} \mid \tau(i) \in R\}$. The robust semantics of any MTL formula φ is defined recursively as follows:

$$[\mathrm{true}]_d(s,i) := +\infty$$

$$[p]_d(s,i) := \mathrm{Dist}_d(s(i), \mathcal{O}(p))$$

$$[\neg \varphi]_d(s,i) := -[\varphi]_d(s,i)$$

$$[\varphi_1 \wedge \varphi_2]_d(s,i) := [\varphi_1]_d(s,i) \sqcap [\varphi_2]_d(s,i)$$

$$[\varphi_1 \mathbf{U}_I \varphi_2]_d(s,i) := \sqcup_{i' \in \tau^{-1}(t +_R I)} ([\varphi_2]_d(s,i') \sqcap \sqcap_{i<i''<i'} [\varphi_1]_d(s,i''))$$

Where $t +_R I = \{t'' \in R \mid \exists t' \in I, t'' = t + t'\}$.

And because $F_I \varphi = \mathrm{true}\, U_I\, \varphi$, we can derive the discrete-time robust semantics for $[F_I \varphi]_d(s,i)$ as follows:

$$[F_I \varphi]_d(s,i) := \sqcup_{i' \in \tau^{-1}(t +_R I)} [\varphi]_d(s,i')$$

2.2.3 Time robustness

In this section we briefly recap time robustness. Originally, a notion of time robustness was introduced in [2] for timed state sequences that are generated by dynamical systems. In this thesis, we use the more general notion of past and future time

robustness as introduced in [9]. Time robustness is introduced in order to quantify the satisfaction problem of signals regarding time and to obtain the characteristic of time-shifting events.

The formal definition of left and right time robustness regarding discrete time is shown below:

$$\theta^{-}(\mathrm{p}, \mathrm{s}, i) = \mathrm{p}[i] \cdot \max\{\mathrm{d} \geq 0 \text{ s.t. } \forall i' \in \mathrm{N} \text{ that } \tau(i') \in [\tau(i) - \mathrm{d}, \tau(i)], \mathrm{p}[i'] = \mathrm{p}[i]\}$$

$$\theta^{+}(\mathrm{p}, \mathrm{s}, i) = \mathrm{p}[i] \cdot \max\{\mathrm{d} \geq 0 \text{ s.t. } \forall i' \in \mathrm{N} \text{ that } \tau(i') \in [\tau(i), \tau(i) + \mathrm{d}], \mathrm{p}[i'] = \mathrm{p}[i]\}$$

Here we defined the time robustness of MTL formula φ regarding a trace s at sampling point i. And then we apply this rule inductively to the rules of MTL formula introduced in section 2.1.

Considering the signal in Fig 2.3, signal1 is 3sin(2t), signal2 is a constant with value is 2.5 and signal3 is 3sin(2t-3.14) which is essentially signal1 shifted right for σ. Suppose we have an MTL formula $\varphi \vDash \mathrm{F}_{[0,1]}\mathrm{p}_1$ where p_1 is x>2. The formula requires that the signal eventually reach a value which exceeds 2 in time 0 to time 1 including time 0 and time 1. If we apply space robustness computation here, we will get that the space robustness degree for signal1 is 1, space robustness degree for signal2 is 0.5 and space robustness degree for signal3 is -2. However, by observation signal2 satisfies the requirement all the time during interval [0, 1] while signal1 fail to meet the requirement about half of the time. To some sense, signal2 is more robustness with regard to time than signal1 over formula φ. Also signal3 is signal1 shifted right and space robustness could not catch this characteristic in this case. Thus, in order to quantify the satisfaction

problem regarding time and capture the effect of shifting events the time robustness is introduced. In this thesis, we implemented a dynamic programming algorithm for time robustness in MATLAB toolbox DP-T-TALIRO.

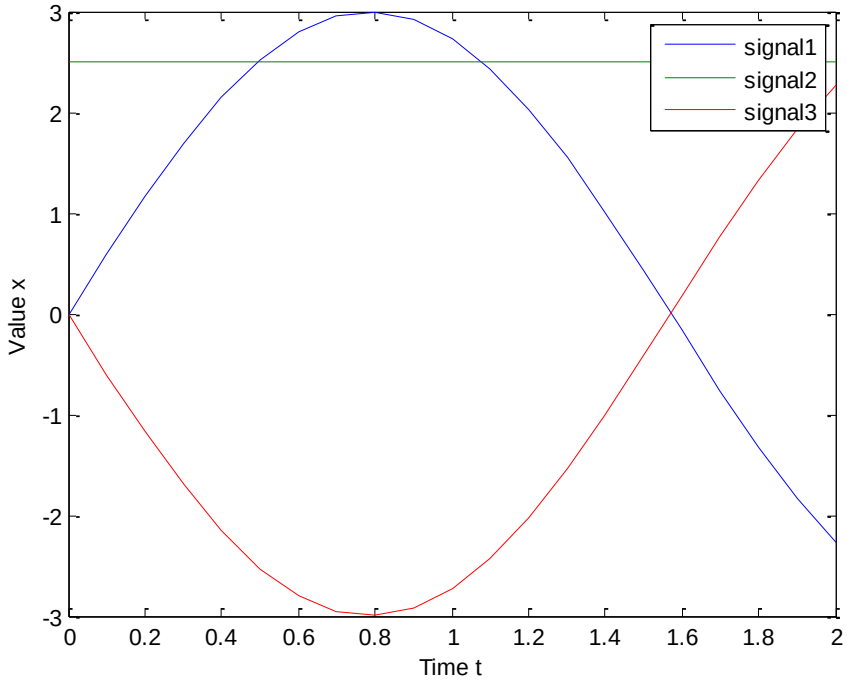

Fig 2.3: Signal1 is 3sin(2t); Signal2 is 2.5; Signal3 is 3sin(2t-3.14)

2.2.4 Polarity

The polarity of a parameter is formally defined in [10]. It is mainly used in parameter estimation problem. Essentially, given a MTL formula φ, if we increase the value of the parameter p and it becomes easier to satisfy the formula then we say the polarity $\pi(p, \varphi)$ is positive. Similarly the polarity is negative if we increase the value of the parameter and it becomes harder to satisfy the MTL formula.

Let ‘+’ and ‘-’ indicate positive and negative polarities respectively. And ‘$\mathbb{T}$’ and ‘$\perp$’ indicate undefined and mixed polarities and c denotes for a constant number. We can inductively define the polarity of a magnitude parameter p in a MTL formula φ as follows:

$$\pi(p, \mathrm{f(x)} < \mathrm{c}) = \pi(p, \mathrm{f(x)} > \mathrm{c}) = \mathbb{T}$$

$$\pi(p, \mathrm{f(x)} < p) = +$$

$$\pi(p, \mathrm{f(x)} > p) = -$$

$$\pi(p, \neg\varphi) = \sim\pi(p, \varphi)$$

$$\pi(p, \varphi_1 \mathrm{U_I}\, \varphi_2) = \pi(p, \varphi_1 \wedge \varphi_2) = \pi(p, \varphi_1) \circ \pi(p, \varphi_2)$$

Operations $\sim$ and $\circ$ are defined as follows:

$\circ$	$\mathbb{T}$	+	-	$\perp$
$\mathbb{T}$	$\mathbb{T}$	+	-	$\perp$
+	+		$\perp$	$\perp$
-	-	$\perp$	-	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

And

	$\sim$
$\mathbb{T}$	$\mathbb{T}$
+	-
-	+
$\perp$	$\perp$

Timing parameters satisfy rules:

$$\pi(s, \varphi_1 \mathrm{U}_{[b,s]}\ \varphi_2) = +$$

$$\pi(s, \varphi_1 \mathrm{U}_{[s,b]}\ \varphi_2) = -$$

We call a formula a fine formula if the polarity of every parameter is either + or -.

2.3 Related work

The concept of robustness interpretation of requirements of MTL formulas form over continuous and hybrid systems is introduced in [2]. It first forwards the concept of ε-ball which is an open ball centered at any point of the trace with radius ε which assist to explain the robustness degree notion. It defined the discrete-time robustness semantics of MTL formulas in a recursive way. Moreover, it defined the procedure of recursively computing robustness degree of an MTL formula over a finite timed state sequence. The algorithm is implemented as a MATLAB toolbox called FW-TALIRO (ForWard algorithm for TemporAl LogIc RObustness) within the software toolbox S-TALIRO [21].

As the name suggests, FW-TALIRO is based on formula rewriting techniques. More details of FW-TALIRO could be found in [2, 18, 19, 20]

One of the works that motivated this thesis is the work done by Rosu and Havelund in [5]. In that paper they presented a dynamic programming algorithm for Linear Temporal Logic (LTL) [7] formula over finite discrete time signals. Given an LTL formula the algorithm tests whether a finite trace satisfies the formula or not. And the algorithm achieved linear running time and constant memory usage depending merely on the size of LTL formula.

In [9], the authors analyzed the behaviors of continuous and hybrid dynamical system admitting uncertain parameters in continuous time and space. They presented several variants of robustness estimation including time robustness and space robustness in order to reason a trajectory satisfying or violating a given requirement over continuous and hybrid systems. In that paper they provided strategies to compute these robustness variants and the sensitivity of them regarding the parameters of the system or the parameters of the formula. The algorithm is implemented in a MATLAB toolbox called Breach [17] which focuses on simulation of temporal logic properties and reachability analysis and parameter synthesis of dynamic systems.

There have been other efforts in computing robustness degrees for MTL. The authors in [6] introduce a different kind of temporal logic to specify properties of a timed state sequence. They introduced the first application of temporal logic offline monitoring to continuous and hybrid systems. The idea of using Metric Temporal Logic as a formal specification to form real-time system is introduced in [3]. It first gives quantitative

temporal properties to a real-time system and then reasons about the system by turning quantitative temporal operators into metric temporal operators. Thati and Rosu in [29] presented a general monitoring algorithm for checking time stamped traces against requirements represented by MTL and sublogics of MTL.

Authors in [18] introduced a framework based on discrete time analysis for testing Metric Interval Temporal Logic (MITL) [28] specifications regarding continuous time signals. And the parametric identification problem is introduced in [10] with the notion of polarity and validity domain as well as the algorithm to compute the validity domains. At the time of this thesis was written it was not possible to experimentally compare the algorithms in [9, 10] with the algorithms presented here. Such comparison will be delegated to future work.

Chapter 3

DP-TALIRO

This chapter provides an overview of the main algorithm of DP-TALIRO toolbox and describes some main new features included in DP-TALIRO.

3.1 DP-TALIRO Overview

We introduce MATLAB toolbox S-TALIRO first to help readers understand what DP-TALIRO does since DP-TALIRO is one of the building blocks of S-TALIRO. S-TALIRO is a tool for the temporal logic falsification problem. It turns the temporal logic falsification problem into an optimization problem by using the temporal logic robustness as a cost function. Using stochastic optimization techniques, it minimizes the temporal logic robustness in order to find counterexamples to the MTL properties.

The overview of the framework of S-TALIRO is shown in Fig 3.1. The dynamical characteristic of the physical system is captured and abstracted in a Simulink/Stateflow model. It outputs traces become one of the inputs of DP-TALIRO, which are then checked against specifications represented by MTL formulas and atomic propositions represented as predicates. DP-TALIRO computes the robustness metric of these traces with respect to the MTL formula. Then, one of the optimization techniques is chosen to let S-TALIRO regulate the physical model. The optimization algorithm aims to determine what output trace of the model must be analyzed next based on the information of robustness metric. S-TALIRO is equipped with Monte Carlo [22], Ant Colony Optimization [23] and other optimization algorithms. The user can implement other stochastic optimization methods as well. In the end, S-TALIRO would output a falsified

trace if one can be found. Otherwise the least robust trace would be outputted if the process time-out and no falsifying traces can be found. The reader is referred to [15, 16, 21, 22] for more details about S-TALIRO.

As we can see in Fig 3.1, the main components of S-TALIRO toolbox are a temporal logic robust computation block and the stochastic optimization algorithm. FW-TALIRO was used as the main temporal logic robust computation block. FW-TALIRO is developed based on formula rewriting techniques and it is suitable for online monitoring. But for offline monitoring it is too slow which makes S-TALIRO almost impossible to use on large-scale traces and MTL formulas. DP-TALIRO is developed for such circumstances to replace FW-TALIRO as the temporal logic robustness computation block of S-TALIRO.

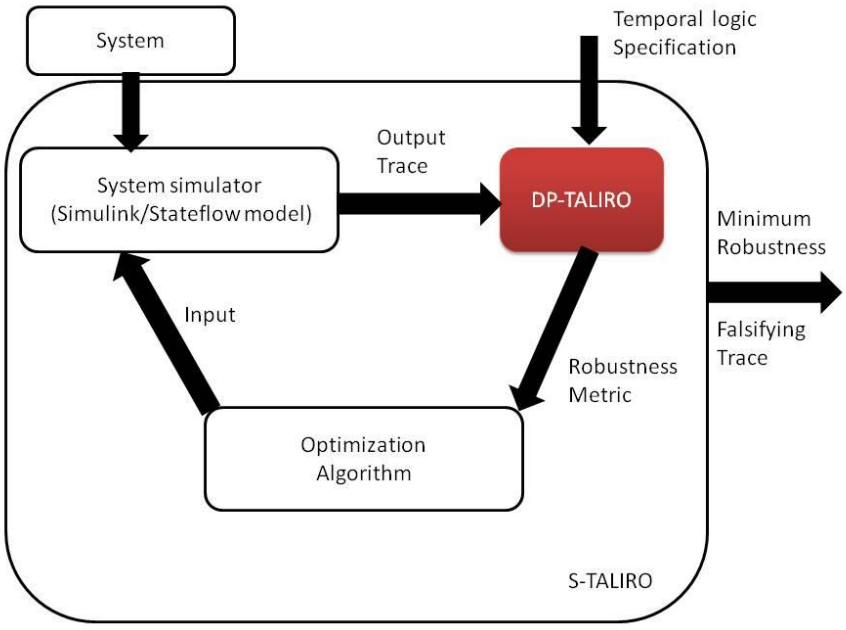

Fig 3.1: overview of S-TALIRO toolbox

3.2 Dynamic programming algorithm of DP-TALIRO

In this section we present the details of the dynamic programming algorithm for computing the space robustness metric of MTL formulas with respect to timed state sequences. We also present details of the implementation of DP-TALIRO.

First, we explain how the dynamic programming algorithm works regarding LTL formulas. In [5], a dynamic programming algorithm is first introduced to test the satisfiability problem of LTL formula with respect to a finite trace of events. This algorithm achieves linear running time and constant memory requirement depending on the size of the LTL formula by visiting the trace of events backwards in time. We develop our algorithm based on this algorithm and expand it for temporal logic robust estimation problem regarding MTL formulas.

First we parse the LTL formula in a tree fashion. We assign subformulas from top down of the parsing tree. The key idea of dynamic programming algorithm for LTL formula is to compute robustness values backwards on the time axis from the last sampling point of the input trace backwards to the first sampling point of the input trace and to compute robustness from bottom up of the LTL formula parsing tree. We store and reuse the temporal logic robustness values of all the subformulas of the sampling point at the very next timed state to compute the temporal logic robustness value of the sampling point at the current timed state. Afterward, robustness values of the sampling point at the current timed state are stored and used to compute the robustness for the previous sampling point while the robustness values stored before are discarded. This process repeats all the way from the last sampling point of the trace to the initial sampling point

of the trace. The robustness of the subformula at root node of the LTL formula parsing tree at the first sampling point of the trace is the robustness value of the LTL formula.

We next present an example to explain how this algorithm works by drawing a dynamic programming table.

Example 3.2.1: Consider the LTL formula $\varphi = G(p_1 \wedge Fp_2)$. We parse this formula as follows in Fig 3.2:

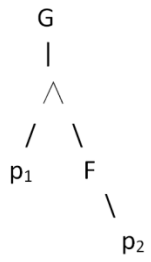

Fig 3.2: Parsing tree of formula $\varphi = G(\mathbf{p_1} \wedge \mathbf{Fp_2})$ of Example 3.2.1

Then we assign subformulas from top down. The corresponding subformulas are:

$\varphi_1 = \varphi = G\varphi_2$

$\varphi_2 = p_1 \wedge Fp_2 = \varphi_4 \wedge \varphi_3$

$\varphi_3 = Fp_2 = F\varphi_5$;

$\varphi_4 = p_1$

$\varphi_5 = p_2$

We draw a dynamic programming table (Table 3.1) to demonstrate how the dynamic programming algorithm works. Each subformula represents a row in the dynamic

programming table and each column represents different sampling point of the input trace. We start from filling the rightmost column which represents the last sampling point of the trace and moving to the left columns one by one.

R[I, J]		J = i	J = i+1
$I = 1$	φ_5	$\mathrm{Disi}_d(\sigma(i), \mathcal{O}(p_2))$	$\mathrm{Disi}_d(\sigma(i+1), \mathcal{O}(p_2))$
$I = 2$	φ_4	$\mathrm{Disi}_d(\sigma(i), \mathcal{O}(p_1))$	$\mathrm{Disi}_d(\sigma(i+1), \mathcal{O}(p_1))$
$I = 3$	φ_3	R[1,i] $\sqcup$ R[3,i+1]	R[1,i+1]
$I = 4$	φ_2	R[2,i] $\sqcap$ R[3,i]	R[2, i+1] $\sqcap$ R[3, i+1]
$I = 5$	φ_1	R[4,i] $\sqcap$ R[5,i+1]	R[4, i+1]

Table 3.1: Dynamic programming table for LTL formula

Here, the trace is $s = (\sigma, \tau)$. $\sigma(t)$ denotes the signal value of time t. Here, we assume time J = i+1 is the last sample of the given trace. Thus, time J = i+1 defines the boundary conditions.

Table 3.1 is populated from top down and from right to left. We fill R[1, i+1] first applying a distance computation of atomic proposition p_2 with respect to the input trace at sample i. Next, we fill R[2, i+1] applying the same semantic for atomic proposition p_1. We fill R[3, i+1], R[4, i+1] and R[5, i+1] in order based on the discrete-time robust semantics defined in Section 2.2.2. Then, we fill R[1,i] to R[5,i] in order applying semantics in Section 2.2.2 using a distance computation if the subformula is an atomic proposition or a supremum/infimum operation or previously computed values. The robustness value for the formula $\varphi = G(p_1 \wedge Fp_2)$ with respect to trace s at the initial time is the value of R[5, i] when i = 1.

The worst case running time of the algorithm above is $O(|\varphi||\tau|)$ which is linear regarding the size of the formula and the length of the trace. This is easy to verify since the number of rows of the dynamic programming table above grows linearly according to the size of the LTL formula and the number of the columns of the table equals to the length of the trace.

Example 3.2.1 illustrates how a dynamic programming algorithm works for LTL formulas. MTL formulas are essentially LTL formulas combined with time constraints and the algorithm is more complicated. So the algorithm for MTL formulas has some similarity with the algorithm for LTL formulas.

First, we parse the MTL formula in a tree fashion. We assign subformulas from top down of the parsing tree. We still compute robustness values backwards in time from the last sampling point to the first sampling point and compute robustness from bottom up of the MTL formula parsing tree. For MTL formulas, we store and reuse the temporal logic robustness values of all the subformulas with indices $i \in \tau^{-1}(t + {}_{R}I)$ in order to compute the temporal logic robustness of the sampling point at the current state. The robustness value of the subformula at root node of the MTL formula parsing tree at the first sampling point of the trace is the robustness value of the MTL formula.

Here we use another example to illustrate dynamic programming algorithm for MTL formulas.

Example 3.2.2: Consider MTL formula $\varphi = F_{[0.3,\ 1.1]}p$.

Atomic proposition: $\mathcal{O}(p) = \{x \in R \mid x>0\}$ and input signal as follows:

t(time)	0	0.2	0.4	0.6	0.8
X(value)	5	4	3	2	1

Table 3.2: Input signal of Example 3.2.2

First we could easily compute the robustness of formula $\varphi = p$ regarding the input signal which is X-0 shown below

t(time)	0	0.2	0.4	0.6	0.8
p	5	4	3	2	1

Table 3.3: Robustness of formula $\varphi = p$ regarding the input signal of Example 3.2.2

And we parse this MTL formula simply as follows:

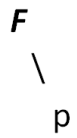

Fig 3.3: Parsing tree of formula $\boldsymbol{\varphi} = F_{[0.3,\ 1.1]}\mathbf{p}$ of Example 3.2.2

We assign the subformulas:

$$\varphi_1 = \varphi = F\varphi_2$$

$$\varphi_2 = p$$

Filling the dynamic programming table for MTL formulas is similar to what we did for the dynamic programming table of LTL formulas. Accord to the algorithm we should use the results in column which indices belongs to $\tau^{-1}(t + 0.3)$ through $\tau^{-1}(t + 1.1)$ to compute the results in column $\tau^{-1}(t)$. Noted that here we use different boundary

conditions according to different temporal logic operators. In this example, the robustness would be set to negative infinity (-∞) if boundary condition is triggered to indicate that the set of indices is an empty set. We fill the dynamic programming table as follows:

i(index)	1	2	3	4	5
t(time)	0	0.2	0.4	0.6	0.8
$\varphi_2 = p$	5	4	3	2	1
$\varphi_1 = \varphi = F_{[0.3,\ 1.1]}p$	3 ⊔ 2 ⊔ 1	2 ⊔ 1	1	-∞	-∞

Table 3.4: Dynamic programming table for MTL formula of Example 3.2.2

As shown in the table above, we start by filling φ at index i=5. We search for the results of subformula φ_2 from time 0.8+0.3 to 0.8+1.1 and the result is the empty set since the timed state sequence is not defined. Thus, we apply the boundary condition and since the temporal logic operator here is 'eventually' operator we set the result as negative infinity. Next, we fill the table at index i=4. We search for the results of subformula φ_2 from time 0.6+0.3 to 0.6+1.1 and again we can find none and we set the robustness as negative infinity. For row φ_1 column i=3, we search for the results of subformula φ_2 in between time interval [0.4+0.3, 0.4+1.1] which is the result in column i=5. We apply the same rule and fill the table all the way down to column t=0 and the result is 3 ⊔ 2 ⊔ 1 which is 3 ultimately. So the robustness of MTL formula $\varphi = F_{[0.3,\ 1.1]}p$ regarding this given input signal is 3.

We can observe from Example 3.2.2 that even though we apply the dynamic programming algorithm for MTL formula we still have to store results of subformula φ_2

in column i=3, i=4 and i=5 in order to compute the result of φ_1 in column i=1. In fact, the columns of results needed to be stored depend on the time constraints of the temporal logic operators. The pseudocode for the dynamic programming algorithm of MTL formulas is presented in Algorithm 3.1 from [15].

Algorithm 3.1 Temporal Logic Robustness Computation

Input: The MTL formula φ , the trace $s = (\sigma, \tau)$, the distance metric d and the observation map $\mathcal{O}$

Output: Return the value stored in s[1,1]

1. **Procedure** DP-TALIRO (φ, $\mathcal{O}$, s, d)
2. **for** j $\leftarrow$ | τ | to 1; **for** i $\leftarrow$ | φ | to 1 **do**
3. **if** $\psi_i = T$ **then** s[i, j] = T
4. **else if** $\psi_i = p$ **then** s[i, j] $\leftarrow$ $Dist_d(\sigma(j), \mathcal{O}(p))$
5. **else if** $\psi_i = \neg \psi_k$ **then** s[i, j] $\leftarrow$ - s[k, j]
6. **else if** $\psi_i = \psi_{k1} \vee \psi_{k2}$ **then**
7. s[i, j] $\leftarrow$ $s[k_1, j] \sqcup s[k_2, j]$
8. **else if** $\psi_i = \psi_{k1} \mathcal{U}_I \psi_{k2}$ **then**
9. **if** j = | τ | **then** s[i, j] $\leftarrow$ $K_\epsilon(0, I) \sqcap s[k_2, j]$
10. **else if** $\mathcal{I} = [0, +\infty)$ **then**
11. s[i, j] $\leftarrow$ $s[k_2, j] \sqcup (s[k_1, j] \sqcap s[i, j+1])$
12. **else**
13. $b_l \leftarrow \min \mathcal{J}(j, I)$;
14. $b_u \leftarrow \max \mathcal{J}(j, I)$;
15. $s_{min} \leftarrow \sqcap_{j \le j' < bl} s[k_1, j']$;
16. s[i, j] $\leftarrow \bot$;
17. **for** j' $\leftarrow$ b_l to b_u **do**
18. s[i, j] $\leftarrow$ $s[i, j] \sqcup (s[k_2, j'] \sqcap s_{min})$;
19. $s_{min} \leftarrow s_{min} \sqcap s[k_1, j']$;
20. **end for**
21. **if** sup I = $+\infty$ **then**
22. s[i, j] $\leftarrow$ $s[i, j] \sqcup (s[k_1, j] \sqcap s[i, j+1])$
23. **end if**
24. **end if**
25. **end if**
26. **else** if $\psi_i = \mathcal{F}_I \psi_{k1}$ **then**
27. **if** j = | τ | **then** s[i, j] $\leftarrow$ $K_\epsilon(0, I) \sqcap s[k_1, j]$
28. **else if** $\mathcal{I} = [0, +\infty)$ **then**
29. s[i, j] $\leftarrow$ $s[k_1, j] \sqcup s[i, j+1]$
30. **else**
31. $b_l \leftarrow \min \mathcal{J}(j, I)$;
32. $b_u \leftarrow \max \mathcal{J}(j, I)$;
33. s[i, j] $\leftarrow \bot$;
34. **for** j' $\leftarrow$ b_l to b_u **do**
35. s[i, j] $\leftarrow$ $s[i, j] \sqcup s[k_1, j']$;
36. **end for**

```
37.                 if sup I = +∞ then
38.                     s[i, j] ← s[k1, bl] ⊔ s[i, bl+1]
39.                 end if
40.             end if
41.         end if
42.     end for
43. end procedure
```

where k, k_1, k_2 > i; K_ϵ (a, A) = T if a ∈ A and ⊥ otherwise; and $\mathcal{J}$(j, I) = $\tau^{-1}((\tau(j) +_R I) \cap (\tau(j+1) +_R I))$ if sup I = +∞ and $\mathcal{J}(j, I) = \tau^{-1}(\tau(j) +_R I)$ otherwise.

The worst case running time of Algorithm 3.1 is $O(|\varphi||\tau|c)$ where c equals to $\max_{0\leq j\leq|\tau|,}([j, \max \mathcal{J}(j, I)])$. Here I is the time constraints of any temporal logic operators in the MTL formula. We already know the worst case running time of dynamic programming algorithm of LTL formula is $O(|\varphi||\tau|)$. And here 'c' stands for the maximum sampling point possible from any point j of the trace up to b_u which is the maximum sampling point allowed according to the time constraints and the structure of the trace. In another word, 'c' is the biggest number of iterations in line 17 and line 34 of Algorithm 3.1.

There are however two hidden factors that would affect the running time of the algorithm above. One is the time cost to compute the distance function which is based on the sets and the structure of state-space and the details are addressed in [2]. Another factor is the time to compute b_l and b_u in line 13 and 14 and line 31 and 32. Since the traces DP-TALIRO deals with can have thousands of sampling points with non-constant steps between any two sampling points, and since for each sampling points b_l and b_u are different, there are thousands of b_l and b_u needed to be calculated. We found that the time computing b_l and b_u grows exponentially when the length of the input trace grows. The

time taken to compute b_l and b_u drastically slows down the computation speed of DP-TALIRO toolbox. In order to address this problem, we propose an algorithm to compute b_l and b_u in linear time. The pseudocode for the time-stamp computation is presented in Algorithm 3.2.

We extract from the input MTL formula and store the higher bound value of a time interval of an MTL subformula in ‘ubd’ and the lower bound value in ‘lbd’. We use ‘Highi’ and ‘Lowi’ to indicate the temporal maximum and minimum index of sampling points thus define the range in which we search for a maximum mapping index with respect to current time stamp and store in b_u and a minimum mapping index and store in b_l.

Algorithm 3.2 Time-stamp Bounds Computation

Input: The trace $s = (\sigma, \tau)$ and the length of the trace $\omega = |\tau|$
Output: Return b_l and b_u

```
1.  b_u = -∞;
2.  b_l = ∞;
3.  for i ← ω to 0
4.     if ubd = +∞ then b_u = ω;
5.     else
6.        if b_u = -∞ then
7.           Highi = ω;
8.        else
9.           Highi = b_u;
10.       Lowi = 0;
11.       end if
12.       for j ← Highi to Lowi
13.          if j ≥ 0
14.             TempU = ubd + time(i);
15.             if time(j) < TempU
16.                b_u = j;
17.                break;
18.             end if
19.          end if
20.       end for
21.    end if
22.       if b_l = ∞ then
23.          Highi = ω;
24.       else
25.          Highi = b_l;
26.       Lowi = Highi – 1;
27.       end if
28.       for j ← Highi to Lowi
29.          if j ≥ 0
30.             TempL = lbd + time(i);
31.             If time(j) > TempL
32.                b_l = j;
33.                Lowi = j-1;
34.             end if
35.          end if
36.       end for
37. end for
```

We compute b_u by finding the first/maximum sampling point which meets the requirement of (j, I) in Algorithm 3.1 and compute b_l by finding the last/minimum

sampling point which meets the requirement of (j, I). It is easy to verify that b_l and b_u is non-increasing every iteration since the trace is monotonic.

Initially we set the value of b_u to negative infinity and the value of b_l to infinity indicating neither b_u nor b_l is set.

To set the value for b_u, we start by searching from the last sampling point backwards in time until the first sampling point we found which time stamp is smaller than 'ubd' plus current time stamp. The index value of this sampling point we found is stored in b_u as the maximum index with respect to the current time stamp and we terminate the '**for**' loop. We search the value for b_u regarding next time stamp by starting from the index last b_u indicates and backwards in time.

To set the value for b_l, we start by searching from the last sampling point backwards in time until the last sampling point we found which time stamp is bigger than 'lbd' plus current time stamp. We search the value for b_l regarding next time stamp by starting from the index last b_l indicates and backwards in time.

The worst case running time of Algorithm 3.2 is $O(2|\tau|)$. We traverse all the sampling points backwards in time once and all the sampling points stored in b_u and b_l once more and make as many comparisons.

Algorithm 3.2 computes from the last sampling point backwards to the first sampling point because Algorithm 3.1 does so and, thus, we can save computation time by executing both algorithms together and traverse the input trace once.

Noted that for an MTL formula or subformula which has the time constraints of [0, ∞), it is actually an LTL formula and DP-TALIRO would treat it as an LTL formula thus saving computation time since computing an LTL formula is faster than computing an MTL formula given other conditions being equal.

3.3 Dynamic programming algorithm for time robustness

We have introduced the dynamic programming algorithm for space robustness computation of MTL formulas. In this section, we apply similar approach for time robustness computation.

In order to compute time robustness, we only need to replace the distance computation in line 4 of Algorithm 3.1 with time-distance computation.

Algorithm 3.3 Time-distance Computation

```
1.  if CurSign = PrevSign
2.      if CurSign
3.          T_rob = |PrevT_rob| + |CurTime – PrevTime|;
4.      else
5.          T_rob = - (|PrevT_rob| + |CurTime – PrevTime|);
6.  else
7.          T_rob = 0;
```

Algorithm 3.3 presents the pseudocode of computing time-distance and itself is computed dynamically. This algorithm could be used for computing both past time distance and future time distance. It keeps updating 'CurSign' and 'PrevSign' by looking up the space robustness value. We illustrate this algorithm by presenting another example.

Example 3.3.1: Consider MTL formula $\varphi = p$ where atomic proposition: $\mathcal{O}(p) = \{x \in R \mid x>0\}$ and input signal as follows:

t(time)	0	0.2	0.4	0.6	0.8
X(value)	3	1	-1	-3	-5

Table 3.5: Input signal of Example 3.3.1

First, we could easily compute the space robustness of formula $\varphi = p$ regarding the input signal which is X-0 shown below:

t(time)	0	0.2	0.4	0.6	0.8
p	3	1	-1	-3	-5

Table 3.6: Robustness of formula $\varphi = p$ regarding the input signal of Example 3.3.1.

And then we apply Algorithm 3.3 and compute the future time robustness and past time robustness:

t(time)	0	0.2	0.4	0.6	0.8
p	3	1	-1	-3	-5
θ^+	0.2	0	-0.4	-0.2	0
θ^-	0	0.2	0	-0.2	-0.4

Table 3.7: Time robustness of formula $\varphi = p$ regarding the input signal of Example 3.3.1

We compute the future time robustness θ^+ from right to left and start with filling the rightmost column and since it is the boundary of the input trace we set the time robustness to 0 to indicate that there are no robustness values in the future which have the same sign as the robustness value at current time. Then, we fill the column t = 0.6, we look up the space robustness of the current column and the previous column and find that they have the same sign because both of them are negative numbers. So we accumulate

the time-distance by adding the time interval which is 0.2 on the previous time robustness which is 0. We multiply the time-distance by -1 which indicates that the space robustness at current time is negative and the result is -0.2. Then, we fill the column t = 0.4 and so on. Similarly we compute the past time robustness θ^{-} from left to right and fill the leftmost column with 0 because of the boundary condition. We fill the column t = 0.2 by adding the time interval which is 0.2 on the previous time robustness which is 0 and multiple 1 and the result is 0.2. Finally we fill the column t = 0.8 with past time robustness -0.4.

Noted that here we set both future time robustness and past time robustness to 0 whenever the signs are different between current column and the previous column. The reason is that somewhere between these two sampling points the trace reach the boundary of satisfying or violating the requirement of the MTL formula. Since there is no way to know where exactly this boundary point lies given a discrete time trace, we choose to set the time robustness to 0 in order to make sure the correctness of the toolbox. Dynamic programming of time robustness is separately implemented as the DP-T-TALIRO toolbox. DP-T-TALIRO is also integrated in S-TALIRO yet it could run alone as a toolbox as well.

3.4 Most related iteration and predicate

DP-TALIRO can return the most related iteration and most related predicate automatically. Here, ‘most related iteration’ means if the robustness value is changed at this specific iteration the output robustness value may be affected as well. Similarly, ‘most related predicate’ means if the robustness value of the specific predicate is changed

the output robustness value will be affected as well. When there is a tie (means several different iterations or different predicates affect the output robustness equally), it will only show one of them.

In this section, we illustrate how the feature 'most related iteration' and 'most related predicate' works. We consider a room with a heater and two sensors. The heater is turned on all the time so that the room temperature increases monotonically. The sensors are set at different threshold values. If the room temperature exceeds the threshold values the sensors will be activated and they will beep.

Example 3.5.1: There is a heater in the room and it is always turned on. The room temperature signal is shown as follows:

Time(t)	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2
Temperature(X)	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2

Table 3.8: Temperature regarding time as input signal of Example 3.5.1.

Atomic proposition: $\mathcal{O}(\text{sensor1}) = \{X \in R \mid X > 4\}$

Atomic proposition: $\mathcal{O}(\text{sensor2}) = \{X \in R \mid X > 3\}$

The MTL formula is shown as follows:

$\varphi = (F_{[0,2]}\text{sensor1}) \vee (F_{[0,2]}\text{sensor2});$

Formula $(F_{[0,2]}\text{sensor1}) \vee (F_{[0,2]}\text{sensor2})$ says that from time 0 to 2 either eventually sensor1 is activated and start to beep or sensor2 is activated and start to beep. Apparently according to the input trace, the maximum environmental temperature it could reach is 2 which is below the threshold values of both sensor1 and sensor2 so neither would be

activated. Here, the ‘most related predicate’ function can provide information on which sensor is closer to be activated and the ‘most related iteration’ function can provide information on which sampling point of the input trace is the most related sensor that is closest to be activated. If we had to consider a bunch of sensors, it would be tedious to solve the same problem manually. Thus, the ‘most related predicate’ and the ‘most related iteration’ function can provide useful information automatically.

For Example 3.5.1 the robustness value is -1, the ‘most related predicate’ is ‘sensor2’ and the ‘most related iteration’ is time t = 2. The robustness value is -1 meaning that the specification (the MTL formula φ) is not met and the distance from meeting the requirement is 1. To be specific, the maximum environmental temperature is 2 at time 2 and in order to meet the specification which is to activate either ‘sensor1’ or ‘sensor2’ the environmental temperature should at least reach 3. The distance between 2 and 3 here is 1. The input trace and atomic propositions are shown in Fig 3.5. The temperature grows as time increases. The last sampling point is clearly the closet to both threshold values of ‘sensor1’ and ‘sensor2’. Thus, it is the most related iteration and ‘sensor2’ is closer to the input trace. Thus it is the most related predicate.

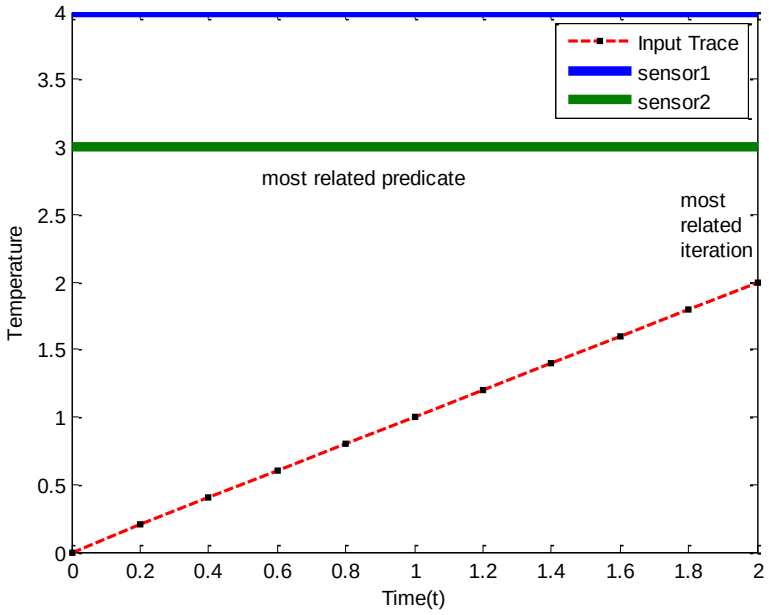

Fig 3.4: most related iteration and predicate result of Example 3.5.1

Chapter 4

EXPERIMENTS AND APPLICATION

4.1 Running time comparison between DP-TALIRO and FW-TALIRO

In this section, we will analyze the running time of DP-TALIRO and compare it with FW-TALIRO over LTL and MTL formulas.

Example 4.1.1: Given the input sequence: Signal = t + 0.5sin(2t)

Atomic proposition $\mathcal{O}(p_1) = \{ x \in R \mid x \geq -2 \}$

Atomic proposition $\mathcal{O}(p_2) = \{ x \in R \mid x \leq 2 \}$

MTL specification to be $\varphi = G\ (F_{[0, 6.28]}(p_2 \wedge F_{[0, 3.14]}\ p_1))$;

Formula φ states that atomic proposition p_2 and eventually p_1 from time 0 to 3.14 hold within time interval [0, 6.28] should happen infinite often.

We use DP-TALIRO and FW-TALIRO to compute the robustness using the setting above regarding the input sequence of different length 1, 5, 10, 20, 40 and 60 respectively and record the computation time in the table below.

Index	Trace Length	DP-TALIRO	FW-TALIRO
1	1	0.003	0.003
2	5	0.003	0.004
3	10	0.003	0.016
4	20	0.003	0.267
5	40	0.003	6.222
6	60	0.004	27.56

Table 4.1: Time comparison between DP-TALIRO and FW-TALIRO

As we can see in Table 4.1, running time of FW-TALIRO grows exponentially and it takes over 25 seconds to compute robustness over a trace with merely 60 sampling points regarding a moderate sized MTL formula. Meanwhile, DP-TALIRO takes no more than 0.005 sec to compute the same robustness.

Example 4.1.2: We consider a more complex model of a powertrain system [30]. The system is modeled in Checkmate [31]. This is the same example as used in [15]. The Stateflow chart for the shift scheduler is shown in Fig 4.1. The system has 3 main components, 6 continuous state variables and 2 Stateflow charts.

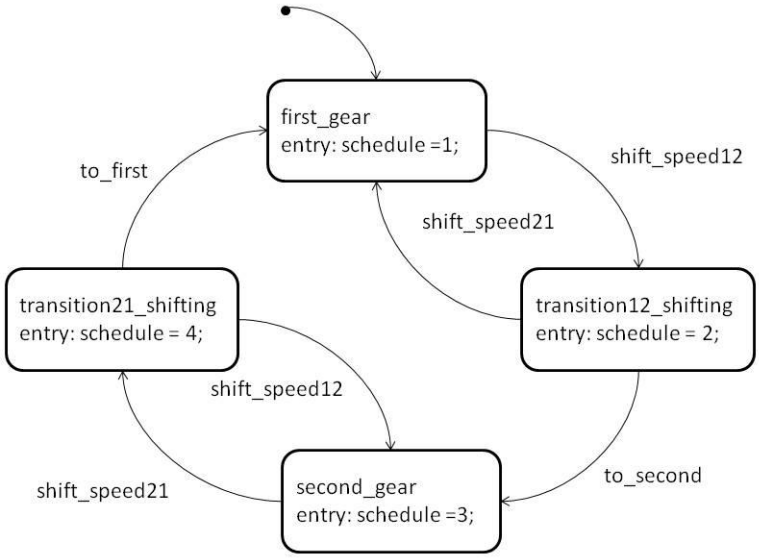

Fig 4.1: The shift scheduler of Example 4.1.2

The challenge problem proposed in [30] is that whether the powertrain system will switch from second gear to first to second from speed 0 to 100km/hr with respect to constant road grade and throttle input. Here, the road grade and throttle position are the initial parameters for the system and constant values of them must be chosen in order to ensure the initial acceleration of the vehicle is greater than zero.

The LTL specification that represents the requirement for gear transition "second to first to second" is shown below. g_1 and g_2 are atomic propositions indicating that the system is in state first_gear and second_gear.

$$\varphi_1 = \neg \mathrm{F}\,(g_2 \wedge \mathrm{F}\,(g_1 \wedge \mathrm{F}\, g_2))$$

Applying S-TALIRO to the above problem we get one trajectory falsify φ_1 when road grade $\approx$ 0.128 and throttle $\approx$ 44.2. S-TALIRO equipped with DP-TALIRO used 1466 simulations and took 49.9 sec altogether. Each robustness computation time is about 0.011 sec.

We also consider a more useful property states that the gear transition from second to first to second should not happen within 2.5 sec. To put another way, whenever the system operates in first gear, then it should not operates in second gear within 2.5 sec. The MTL specification that represents this requirement is:

$$\varphi_2 = \mathrm{G}\left((\neg\, \mathrm{g}_1 \wedge \mathrm{X}\, \mathrm{g}_1) \rightarrow \mathrm{G}_{[0,\ 2.5]} \neg\, \mathrm{g}_2\right)$$

Another useful property to be considered for powertrain systems is to verify that the oscillation between gears is within acceptable limits. Such as, whenever the system is in transition from gear 2 to gear 1, then the derivative of the torque β is under certain limits. The LTL specification that captures this requirement is:

$$\varphi_3 = \mathrm{G}\,(\mathrm{g}_{21} \rightarrow \mathrm{b})$$

Where $\mathcal{O}(\mathrm{b}) = \{\beta \in \mathrm{R} \mid \beta \leq 450\}$.

We compare average computation time of robustness with respect to $\varphi_1, \varphi_2, \varphi_3$ between DP-TALIRO and FW-TALIRO as shown in Table 4.2. We remark that we did following tests on an Intel Core Duo at 2.10GHz with 3.00 GB RAM and Windows Vista 32-bit operating system.

Spec.	DP-TALIRO(sec)	FW-TALIRO(sec)
φ_1	0.011	60<
φ_2	0.036	0.067
φ_3	0.018	0.009

Table 4.2: Comparison of DP-TALIRO and FW-TALIRO of Example 4.1.2

We can see that robustness computation time of DP-TALIRO is acceptable for all three formulas. But in certain circumstances such as formula φ_3 in this example, FW-TALIRO outperforms DP-TALIRO.

4.2 Running time analysis of DP-TALIRO

In this section, we test the ability of DP-TALIRO of handling the large scale of data required by S-TALIRO. We use the same trace and MTL formula as in Example 4.1.1 and increase the trace length to test the performance of DP-TALIRO.

Trace Length	DP-TALIRO
100	0.004
600	0.006
3,600	0.021
21,600	0.113
129,600	0.642

Table 4.3: Running time of DP-TALIRO regarding different lengths of input trace

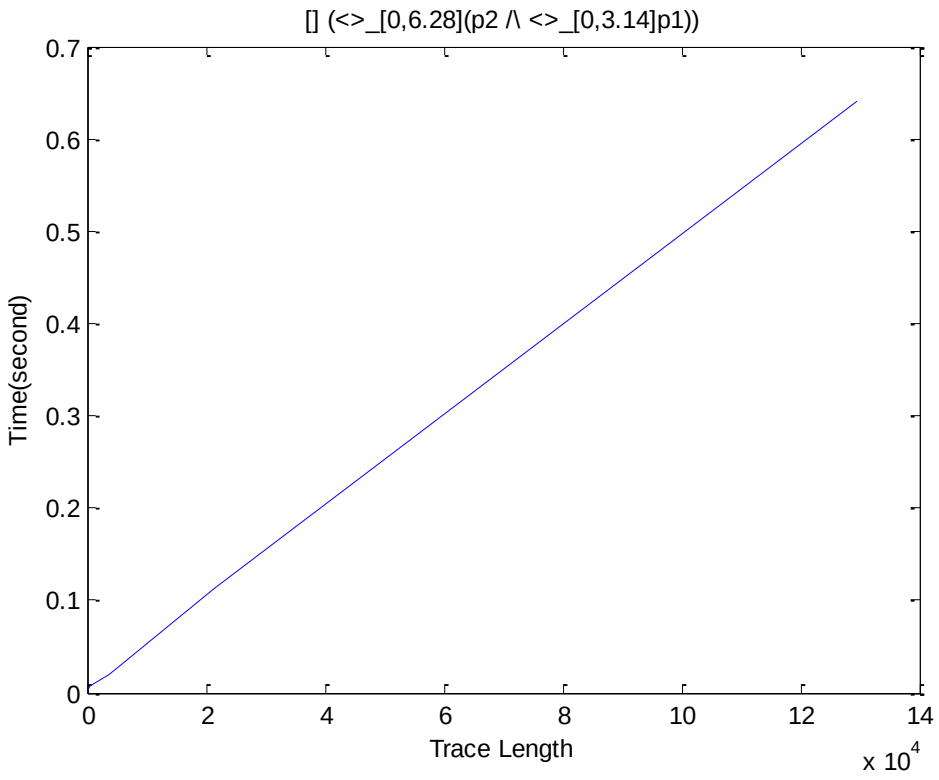

Fig 4.2: Running time of DP-TALIRO regarding large numbers of sampling points

It takes less than 0.7 sec for DP-TALIRO to compute the robustness over 129600 sampling points and the running time grows almost linearly as shown in Fig 4.2.

Next we test the running time of DP-TALIRO regarding different MTL formulas. Here, we still use the same input trace with 129600 sampling points and test the running time regarding 25 different MTL and LTL formulas. The result is shown in Table 4.4.

Index	Formula:	Time:
1	$G_{[0,1]}\ p_1$	0.208
2	$G_{(3.14,\infty)}\ p_1$	0.213
3	$G\,(p_1 \rightarrow F\ \neg\ p_2)$	0.408
4	$G\,(F_{[0,6.28]}(\ p_2 \wedge F_{[0,3.14]}\ p_1))$	0.719
5	$p_2 \wedge\ p_1$	0.178
6	$p_2 \vee\ p_1$	0.174
7	$F\ p_1$	0.174
8	$G\ p_2$	0.165
9	$X\ p_1$	0.126
10	$G_{[0,1]}\ p_1 \vee X\ p_2$	0.140
11	$p_1\ U\ p_2$	0.173
12	$p_1\ R\ p_2$	0.172
13	$p_1\ U\ p_2\ U\ p_1$	0.276
14	$p_1\ R\ p_2\ R\ p_1$	0.280
15	$p_1\ U\ p_2\ R\ p_1 U\ p_2$	0.390
16	$p_1\ U_{[0.1,2]}\ p_2\ R_{[0.1,2]}\ p_1\ U_{[0.1,2]}\ p_2$	0.653
17	$F_{[0.1,3]}\ p_1$	0.254
18	$(G\ p_1 \wedge F_{[21.9911,\infty)}\ p_2)$	0.363
19	$G\,(p_1 \rightarrow F_{(0,1)}\ \neg\ p_1)$	0.465
20	$G\,(p_1 \rightarrow F_{(0,5)}\,(G_{(0,10)}\ \neg\ p_1))$	0.898

21	$G\,(F\,(p_2 \wedge F\ p_1))$	0.428
22	$F\ p_1 \wedge G\,(p_1 \rightarrow F\ p_2)$	0.603
23	$G\,(p_1 \rightarrow F\ \neg\, p_1)$	0.452
24	$G\,(\neg\, p_1 \vee F\,(G\ \neg\, p_1)\ p_1)$	0.513
25	$(G_{(0,10)}(\neg\, p_1 \vee F_{[10,20]}\, G\ \neg\, p_1)) \wedge p_1\ U_{[0,20]}\ p_2\ R_{[20,30]}\ p_1\ U\ p_2$	2.864

Table 4.4: Running time of DP-TALIRO with respect to 129600 sampling points

As we can see the running time regarding most formulas is less than 1 second with one exception which is formula 25. Formula 25 is a complex MTL formula and it takes some extra time to compute the robustness. However, even for formula 25, the running time might still be acceptable for certain applications. We have experimentally demonstrated that timing constrains and multiple nested temporal operators have an impact on the robustness computation time. The experimental analysis agrees with the theoretical time complexity analysis in Section 3.3.

Chapter 5

DP-TALIRO APPLICATION

In this section, we present an application of DP-TALIRO as part of S-TALIRO on a parameter estimation problem. We demonstrate DP-TALIRO on the example from [11]. The problem defined in [11] states that given a hybrid system and an MTL formula with one unknown parameter in a predefined range, find the optimal range of the parameter of time that makes the hybrid system violate all the MTL formulas with the values of the parameter in the resulting range. S-TALIRO relies on the temporal logic robustness computation function of DP-TALIRO and turns the parameter estimation problem into an optimization problem using falsification methods and stochastic search methods.

We consider the motivation problem in Example 1.1.1. Its Stateflow chart is shown in Fig 5.1.

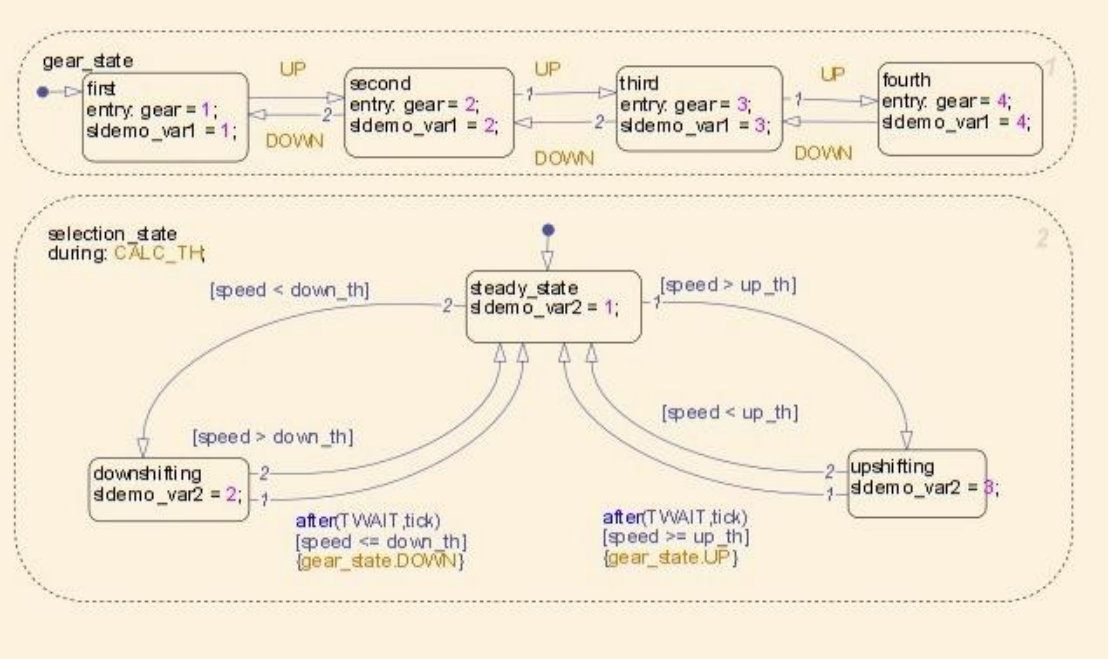

Fig 5.1: Finite State Machine for the automatic drivetrain in Example 4.3.1

We can use S-TALIRO which equipped with DP-TALIRO to solve problems such as "How quick we can reach and exceed 3250 RPM" or "What is the maximum time that the RPM ω cannot exceed 4500 whatsoever". We can write the specification for the problem of "What is the maximum time that the RPM ω cannot exceed 4500 whatsoever" as $\varphi[\theta] = G_{[0,\theta]} \neg p$ where p is ($\omega > 4500$). The robustness of this specification as a function of θ and the input which is the throttle schedule μ is shown in Fig 5.2. The parameter starts from 0 second to 30 seconds and the throttle schedule span from 0 per cent to 100 percent.

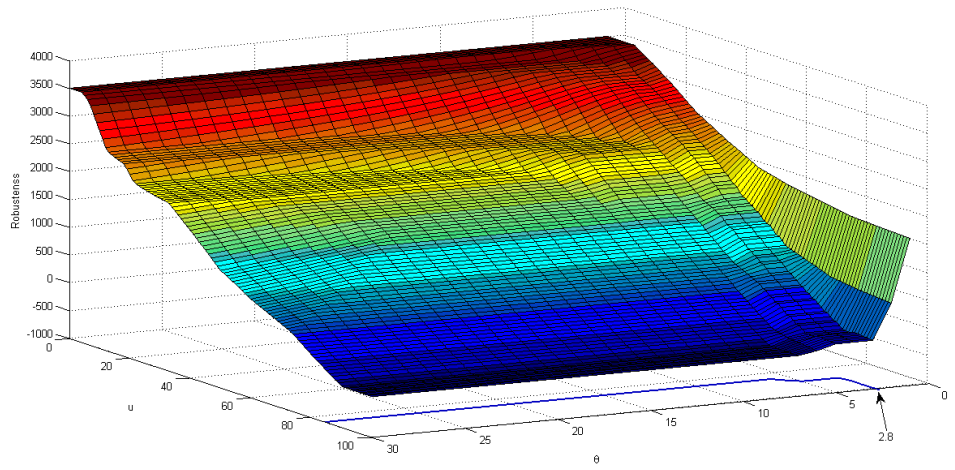

Fig 5.2: Robustness as a function of parameter θ and input μ in Example 4.3.1

The boundary values of parameter θ and input μ which make the robustness 0-that RPM ω equals to 4500-is shown by the blue contour under the surface and we can infer from the graph that $\theta \approx 2.8$. Thus, we say that for any $\theta \in [2.8, 30]$, $\varphi[\theta] < 0$ regarding this model.

Chapter 6

CONCLUSION AND FUTURE WORK

We think MTL is a promising approach for formalizing system requirements of embedded control software. Thus, in this thesis, we have presented a toolbox DP-TALIRO which is based on dynamic programming algorithm for computing temporal logic robustness for MTL specifications. We have demonstrated that DP-TALIRO has much improved performance over its ancestor FW-TALIRO. The experiments show that DP-TALIRO has an approximated linear running time regarding LTL and MTL specifications. Also we have integrated new features to increase the flexibility and usability of DP-TALIRO.

There are several new directions worth exploring further. Currently DP-TALIRO only supports future time temporal logic operators. In the future, we would like to support past time temporal logic [14, 24] by including past time temporal logic operators such as ‘since’, ‘sometime in the past’ and ‘always in the past’. In order to further increase the running speed, more work could be done by letting DP-TALIRO automatically detect and omit irrelevant parts of the input trace and adjust the length of the input trace.

REFERENCES

[1] R. Alur and T. Henzinger. Real time logics: complexity and expressiveness. In Fifth annual symposium on logic in computer science, pages 390-401. IEEE Computer Society Press, 1990.

[2] G. E. Fainekos and G. J. Pappas, Robustness of temporal logic specifications for continuous-time signals, Theoretical Computer Science, vol. 410, no. 42, pp. 4262–4291, 2009.

[3] Ron Koymans. Specifying real-time properties with metric temporal logic. Real-Time System., 2(4):255–299, 1990.

[4] A. K. Seda and P. Hitzler, "Generalized distance functions in the theory of computation," The Computer Journal, vol. 53, no. 4, pp. 443-464, 2010

[5] G. Roşu and K. Havelund. Synthesizing Dynamic Programming Algorithms from LinearTemporal Logic Formulae. RIACS Technical report, January 2001.

[6] Oded Maler, Dejan Nickovic: Monitoring Temporal Properties of Continuous Signals. FORMATS/FTRTFT 2004: 152-166

[7] Amir Pnueli, The temporal logic of programs. Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS), 1977, 46–57.

[8] Baber, R. L., The Ariane 5 Explosion: A Software Engineer's View, Technical Report, Computer Science, University of the Witwatersrand, South Africa, February 1997

[9] A. Donze and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In K. Chatterjee and T. A. Henzinger, editors, FORMATS, volume 6246 of Lecture Notes in Computer Science, pages 92-106. Springer, 2010.

[10] E. Asarin, A. Donzé, O. Maler and D. Nickovic. Parametric Identification of Temporal Properties, In: Runtime Verification. Volume 7186 of LNCS., Springer (2012) 147-160

[11] Hengyi Yang, Bardh Hoxha, Georgios E. Fainekos: Querying Parametric Temporal Logic Properties on Embedded Systems. ICTSS 2012: 136-151

[12] Abbas, H., Fainekos, G.E., Sankaranarayanan, S., Ivancic, F., Gupta, A.: Probabilistic temporal logic falsification of cyber-physical systems. ACM Transactions on Embedded Computing Systems (In Press) (2011).

[13] Zhao, Q., Krogh, B.H., Hubbard, P.: Generating test inputs for embedded control systems. IEEE Control Systems Magazine August (2003) 49-57

[14] A. Cimatti, M. Roveri, and D. Sheridan. Bounded verification of Past LTL. In Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD), volume 3312 of *LNCS*, pages 245-259. Springer-Verlag, 2004.

[15] Georgios Fainekos, Sriram Sankaranarayanan, Koichi Ueda and Hakan Yazarel Verification of Automotive Control Applications using S-TaLiRo American Control Conference, Montreal, Canada, June 2012

[16] Sriram Sankaranarayanan and Georgios Fainekos, Falsification of Temporal Properties of Hybrid Systems Using the Cross-Entropy Method, ACM International Conference on Hybrid Systems: Computation and Control, Beijing, China, Apr. 2012

[17] A. Donze. Breach: A Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In Computer-Aided Verification, pages 167-170, 2010.

[18] Georgios E. Fainekos and George J. Pappas, Robust Sampling for MITL Specifications, In the 5th Inter. Conference on Formal Modeling and Analysis of Timed Systems, Salzburg, Austria, October 2007

[19] Georgios E. Fainekos and George J. Pappas, Robustness of Temporal Logic Specifications, In the Workshop on Formal Approaches to Testing and Runtime Verification, Seattle, USA, August 2006

[20] Georgios E. Fainekos and George J. Pappas, Robustness of Temporal Logic Specifications for Finite State Sequences in Metric Spaces, Technical Report MS-CIS-06-05, Department of CIS, University of Pennsylvania, May 2006

[21] Y. S. R. Annapureddy, C. Liu, G. E. Fainekos and S. Sankaranarayanan, S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems, In the Proc. of Tools and algorithms for the construction and analysis of systems, Saarbrucken, Germany, March 2011

[22] T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivancic, A. Gupta and G. Pappas, Monte-Carlo Techniques for Falsification of Temporal Properties of Non-Linear Systems, Hybrid Systems: Computation and Control, Stockholm, Sweden, Apr. 2010

[23] Yashwanth Singh Rahul Annapureddy and Georgios E. Fainekos, Ant Colonies for Temporal Logic Falsification of Hybrid Systems, In the Proceedings of the 36th Annual Conference of IEEE Industrial Electronics, Phoenix, AZ, Nov. 2010

[24] M. Benedetti and A. Cimatti. Bounded model checking for past LTL. In Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS'03, Lecture Notes in Computer Science, Warsaw, Poland, April 2003. Springer-Verlag.

[25] Edmund M. Clarke, Jr., Orna Grumberg and Doron A. Peled, Model Checking, MIT Press, 1999, ISBN 0-262-03270-8.

[26] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, The algorithmic analysis of hybrid systems, Theoretical Computer Science 138 (1) (1995) 3-34.

[27] TaLiRo Tools. [Online]. Available: https://sites.google.com/a/asu.edu/s-taliro/

[28] R. Alur, T. Feder and T. A. Henzinger. The benefits of relaxing punctuality. Journal of the ACM, 43:116–146, 1996

[29] Prasanna Thati, Grigore Roşu, Monitoring Algorithms for Metric Temporal Logic Specifications, Electronic Notes in Theoretical Computer Science (ENTCS), 113, p.145-162, January, 2005

[30] A. Chutinan and K. R. Butts, "Dynamic analysis of hybrid system models for design validation," Ford Motor Company, Tech. Rep., 2002.

[31] B. I. Silva and B. H. Krogh, "Formal verification of hybrid systems using CheckMate: a case study," in Proceedings of the American Control Conference, vol. 3, Jun. 2000, pp. 1679 – 1683.

APPENDIX A

DP-TALIRO USER GUIDE

DP-TALIRO is a tool that computes the robustness estimate of a propositional temporal logic specification with respect to a finite timed state sequence. DP-TALIRO stands for dynamic programming temporal logic robustness computation engine. DP-TALIRO is implemented in MATLAB C (MEX) with dynamic programming based algorithm for both Linear Temporal Logic (LTL) and Metric Temporal Logic (MTL) specifications. The specification is an LTL formula when there is no temporal operator with timing constraints. The time and memory requirements of such formulas are linear with respect to the size of the formula and the input signal. For MTL formulas, the time and memory requirements also depend on the real time constraints.

Version 1.1 supports multi-dimensional signals and time parameter as an input with specified parameter value or range. The time parameter is used in time constraints to provide more flexibility. DP-TALIRO version 1.1 can also output the most related iteration and the most related predicate, as well. Here, the most related iteration means that if the robustness value is changed at this specific iteration the output robustness value could be affected as well. Similarly, the most related predicate means if the robustness value of the specific predicate is changed the output robustness value would be affected as well. When there is a tie i.e., several different iterations or different predicates affect the output robustness, it will only show one of them. DP-TALIRO is integrated into S-TALIRO but it can still be run as a stand along tool.

In this section, we describe the use of function DP-TALIRO in MATLAB. To compile and set up the MATLAB path to DP-TALIRO, one could run `setup-dp-taliro` or `setup_staliro`.

The use interface is as follows:

```
[rob_dp,aux] = dp_taliro(phi,Pred,seqS,seqT,seqL,A,G)
```

Or

```
rob_dp = dp_taliro(phi,Pred,seqS,seqT,seqL,A,G)
```

User can decide whether they want to output auxiliary information (most related iteration and most related predicate) or not.

<table>
<tr><td colspan="3">Output arguments</td></tr>
<tr><td>rob</td><td colspan="2">The robustness estimate. This is a double precision floating point number in case continuous system trajectories or a HyDis object for hybrid system trajectory robustness. To get the continuous state robustness type get(rob,2).</td></tr>
<tr><td>aux</td><td colspan="2">A structure that contains information on the most related iteration and most related predicate.
aux.i indicates the most related iteration for robustness
aux.pred indicates the most related predicate index of the robustness</td></tr>
<tr><td colspan="3">Input arguments</td></tr>
<tr><td rowspan="12">phi</td><td colspan="2">An MTL or LTL formula. The following indicates the correspondence between the symbols of the logic operators and the input ASCII characters and how to define the timing constraints on the temporal operators. If there are no timing constraints following any temporal operators then it is an LTL formula.</td></tr>
<tr><td colspan="2">Syntax:
phi := p | (phi) | !phi | phi \/ phi | phi /\ phi | phi -> phi | phi <-> phi |
| X_{a,b} phi | phi U_{a,b} phi | phi R_{a,b} phi |
| <>_{a,b} phi | []_{a,b} phi</td></tr>
<tr><td>p</td><td>a predicate (it can be any lowercase string)</td></tr>
<tr><td>!</td><td>‘not’</td></tr>
<tr><td>\/</td><td>'or'</td></tr>
<tr><td>/\</td><td>'and'</td></tr>
<tr><td>-></td><td>'implies'</td></tr>
<tr><td><-></td><td>'if and only if'</td></tr>
<tr><td>{a,b}</td><td>where { is [or (, and } is] or) is for defining open or closed timing bounds on the temporal operators.
For example, {a,b} can be [0,1] or (1,2]</td></tr>
<tr><td>X_{a,b}</td><td>the 'next' operator with time bounds {a,b}. It means that the next event should occur within time {a,b} from the current event. If timing constraints are not needed, then simply use X.</td></tr>
<tr><td>U_{a,b}</td><td>the 'until' operator with time bounds {a,b}. If no time bounds are required, then use U.</td></tr>
<tr><td>R_{a,b}</td><td>the 'release' operator with time bounds {a,b}. If no time bounds are required, then use R.</td></tr>
<tr><td><>_{a,b}</td><td>the 'eventually' operator with time bounds {a,b}. If no timining constraints are required, then simply use <>.</td></tr>
<tr><td>[]_{a,b}</td><td>the 'always' operator with time bounds {a,b}. If no timining constraints are required, then simply use [].</td></tr>
</table>

<table>
<tr><td></td><td colspan="2">Examples:
* Always 'a' implies eventually 'b' within 1 time unit:
phi = '[](a -> <>_[0,1] b)';
* a is true until b becomes true after 4 and before 7.5 time units:
phi = 'a U_(4,7.5) b';</td></tr>
<tr><td rowspan="8">Pred</td><td>Pred(i).str</td><td>the predicate name as a string</td></tr>
<tr><td>Pred(i).A,
Pred(i).b</td><td>a constraint of the form Ax<=b</td></tr>
<tr><td>Pred(i).loc</td><td>a vector with the control locations on which the predicate should hold in case of trajectories of hybrid systems. If the control location vector is empty, then the predicate should hold in any location,</td></tr>
<tr><td>Pred(i).par</td><td>the timing parameter name (aka. parameter), one predicate could only have either ‘str’ field or ‘par’ field. Meaning it could either be a traditional predicate or a timing parameter.[1]</td></tr>
<tr><td>Pred(i).value</td><td>the value of the parameter</td></tr>
<tr><td>Pred(i).range</td><td>search range of a parameter[2]</td></tr>
<tr><td colspan="2">Examples:
Define a predicate ‘p1’ x ≤ 1.5 hold in control location 1 or 2:
Pred(1).str = 'p1';
Pred(1).A = 1;
Pred(1).b = 1.5;
Pred(i).loc = [1,2]
Define a parameter ‘t’ with value 2.5 in formula phi = 'F_(t,7.5) p1':
Pred(2).par = 's';
Pred(2).value = 2.5;</td></tr>
<tr><td rowspan="2">seqS</td><td colspan="2">The sequence of states from a Euclidean space X. Each row must be a different sampling instance and each column a different dimension in the state space.</td></tr>
<tr><td colspan="2">For example, a 2D signal sampled at 3 time instances is:
seqS = [0.1 0.2; 0.15 0.19; 0.14 0.18];</td></tr>
<tr><td rowspan="2">seqT</td><td colspan="2">The time-stamps of the trace. It must be a column vector.</td></tr>
<tr><td colspan="2">For example:
seqT = [0 0.1 0.2]';
It should be a monotonically increasing sequence.
Enter [] or ignore if you are interested only about LTL properties.</td></tr>
</table>

[1] Note that a parameter is not a field of predicate but a different type of predicate. We include parameter in the predicate as a special type in order to keep the interface unchanged for better compatibility.

[2] If a parameter has both ‘value’ and ‘range’ field, the ‘range’ field would be omitted and the specific parameter would have a certain value instead of a defined range.

seqL	This is the sequence of locations in case of hybrid system trajectory. It is assumed that each location has a unique numerical (integer) value. It can be omitted in case the predicates refer to global conditions on the continuous state space.
CLG	The control location graph. This is the adjacency matrix or graph of the control locations of the Hybrid Automaton. It can be omitted in case the predicates refer to global conditions on the continuous state space.
GRD	Guard set for each edge of the CLG. For each edge (i,j) of CLG, the set that enables the transition is a polytope of the form Ax <= b. This is a 2D array of stractures: GRD(i,j).A GRD(i,j).b

To setup DP-TaLiRo run `setup_dp_taliro`.

For the hybrid distance metric with distances to the location guards the

Matlab package MatlabBGL is required:

http://www.mathworks.com/matlabcentral/fileexchange/10922

SVN repository for the current version:

https://subversion.assembla.com/svn/s_taliro/truck/dp_taliro

License:

along with this program; if not, write to the Free Software

Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

APPENDIX B

DP-T-TALIRO USER GUIDE

DP-T-TALIRO is a tool that computes the time robustness estimate of a propositional temporal logic specification with respect to a finite timed state sequence. It is developed based on DP-TALIRO. DP-T-TALIRO is implemented in MATLAB C (MEX) with dynamic programming based algorithm for both Linear Temporal Logic (LTL) and Metric Temporal Logic (MTL) specifications. The specification is an LTL formula when there is no temporal operator with timing constraints. Version 1.0 supports multi-dimensional signals. DP-T-TALIRO is integrated into S-TALIRO but it can still be run as a stand along tool.

In this section, we describe the use of function DP-T-TALIRO in MATLAB. To compile and set up the MATLAB path to DP-T-TALIRO, one could run `setup-dp-t-taliro` or `setup_staliro`.

The use interface is as follows:

```
rob = dp_t_taliro(phi,Pred,seqS,seqT,seqL,A,G)
```

<table>
<tr><td colspan="3">Output arguments</td></tr>
<tr><td>rob</td><td colspan="2">The robustness estimate. This is a structure consist of two floating point numbers indicating past time and future time robustness. To get the future time robustness type rob.ft. To get the past time robustness type rob.pt. .</td></tr>
<tr><td colspan="3">Input arguments</td></tr>
<tr><td rowspan="15">phi</td><td colspan="2">An MTL or LTL formula. The following indicates the correspondence between the symbols of the logic operators and the input ASCII characters and how to define the timing constraints on the temporal operators. If there are no timing constraints following any temporal operators then it is an LTL formula.</td></tr>
<tr><td colspan="2">Syntax:
phi := p | (phi) | !phi | phi \/ phi | phi /\ phi | phi -> phi | phi <-> phi |
| X_{a,b} phi | phi U_{a,b} phi | phi R_{a,b} phi |
| <>_{a,b} phi | []_{a,b} phi</td></tr>
<tr><td>p</td><td>a predicate (it can be any lowercase string)</td></tr>
<tr><td>!</td><td>‘not’</td></tr>
<tr><td>\/</td><td>'or'</td></tr>
<tr><td>/\</td><td>'and'</td></tr>
<tr><td>-></td><td>'implies'</td></tr>
<tr><td><-></td><td>'if and only if'</td></tr>
<tr><td>{a,b}</td><td>where { is [or (, and } is] or) is for defining open or closed timing bounds on the temporal operators.
For example, {a,b} can be [0,1] or (1,2]</td></tr>
<tr><td>X_{a,b}</td><td>the 'next' operator with time bounds {a,b}. It means that the next event should occur within time {a,b} from the current event. If timing constraints are not needed, then simply use X.</td></tr>
<tr><td>U_{a,b}</td><td>the 'until' operator with time bounds {a,b}. If no time bounds are required, then use U.</td></tr>
<tr><td>R_{a,b}</td><td>the 'release' operator with time bounds {a,b}. If no time bounds are required, then use R.</td></tr>
<tr><td><>_{a,b}</td><td>the 'eventually' operator with time bounds {a,b}. If no timining constraints are required, then simply use <>.</td></tr>
<tr><td>[]_{a,b}</td><td>the 'always' operator with time bounds {a,b}. If no timining constraints are required, then simply use [].</td></tr>
<tr><td colspan="2">Examples:
* Always 'a' implies eventually 'b' within 1 time unit:
phi = '[](a -> <>_[0,1] b)';
* a is true until b becomes true after 4 and before 7.5 time units:
phi = 'a U_(4,7.5) b';</td></tr>
<tr><td>Pred</td><td>Pred(i).str</td><td>the predicate name as a string</td></tr>
</table>

	Pred(i).A, Pred(i).b	a constraint of the form Ax<=b
	Pred(i).loc	a vector with the control locations on which the predicate should hold in case of trajectories of hybrid systems. If the control location vector is empty, then the predicate should hold in any location,
	Pred(i).par	the timing parameter name (aka. parameter), one predicate could only have either 'str' field or 'par' field. Meaning it could either be a traditional predicate or a timing parameter.[3]
	Pred(i).value	the value of the parameter
	Pred(i).range	search range of a parameter[4]
	Examples: Define a predicate 'p1' x $\leq$ 1.5 hold in control location 1 or 2: Pred(1).str = 'p1'; Pred(1).A = 1; Pred(1).b = 1.5; Pred(i).loc = [1,2] Define a parameter 't' with value 2.5 in formula phi = 'F_(t,7.5) p1': Pred(2).par = 's'; Pred(2).value = 2.5;	
seqS	The sequence of states from a Euclidean space X. Each row must be a different sampling instance and each column a different dimension in the state space.	
	For example, a 2D signal sampled at 3 time instances is: seqS = [0.1 0.2; 0.15 0.19; 0.14 0.18];	
seqT	The time-stamps of the trace. It must be a column vector.	
	For example: seqT = [0 0.1 0.2]'; It should be a monotonically increasing sequence. Enter [] or ignore if you are interested only about LTL properties.	
seqL	This is the sequence of locations in case of hybrid system trajectory. It is assumed that each location has a unique numerical (integer) value. It can be omitted in case the predicates refer to global conditions on the continuous state space.	
CLG	The control location graph. This is the adjacency matrix or graph of the control locations of the Hybrid Automaton. It can be omitted in case the predicates refer to global conditions on the continuous state space.	

[3] Note that a parameter is not a field of predicate but a different type of predicate. We include parameter in the predicate as a special type in order to keep the interface unchanged for better compatibility.

[4] If a parameter has both 'value' and 'range' field, the 'range' field would be omitted and the specific parameter would have a certain value instead of a defined range.

GRD	Guard set for each edge of the CLG. For each edge (i,j) of CLG, the set that enables the transition is a polytope of the form Ax <= b. This is a 2D array of structures: GRD(i,j).A GRD(i,j).b

To setup DP-T-TaLiRo run `setup_dp_t_taliro`.

For the hybrid distance metric with distances to the location guards the

Matlab package MatlabBGL is required:

http://www.mathworks.com/matlabcentral/fileexchange/10922

SVN repository for the current version:

https://subversion.assembla.com/svn/s_taliro/truck/dp_t_taliro

License:

along with this program; if not, write to the Free Software

Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA